(NeXT Tip #39) I/O redirection in csh

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Mon, 13 Sep 1993 10:22:48 -0700 (PDT)

The issue of redirecting 'stderr' independently of 'stdout' came up last week on the MSOB-Help list. Before I get into how to do it, let's review I/O redirection in general. We all (hopefully) know basic redirection in 'csh':

program < file -- use 'file' as the standard input to 'program'
program > file -- use 'file' as the standard output for 'program'

Of course these can be used together and there are many variations:

program >> file -- append to the end of 'file' as the standard output
program << word -- read input up to 'word'

The '<<' operator is used primarily in scripts, here's an example:

```
if ( $#argv == 0 ) then
    cat << EOT; exit 0
Usage background -reset
    background -gray GrayShade
    background -{file,center,scale,minscale,tile,debug} FileName

Options can be shortened to just their first letter, eg. 'background -g 0.666'
The 'gray' option can also be combined with 'file', 'center' and 'minscale'
EOT
endif
```

This script fragment prints out some usage information and exits if there weren't any arguments given. This could have been done with multiple 'echo' calls but this approach is cleaner if you don't need to do variable substitution. There are other situations where '<<' is useful in scripts.

But back to redirection variations:

program >! file -- forcibly use 'file' as the standard output
program >>! file -- forcibly append output to the end of 'file'

These minor variations are rarely used and the only difference is that they ignore the setting of the 'noclobber' variable. (The 'noclobber' variable is set in our local default .cshrc if the TYRO variable is set.) The 'noclobber' variable can be used to keep '>' from destroying existing files and keep '>>' *>From appending to files that don't already exist.*

These variations should be used, for example in scripts that append to scratch files on /tmp that might not already exist. (Doing your first write to the scratch file with '>' is an alternate solution but sometimes you don't know where the first write to the file will occur in your script.)

But again, back to the redirection variations:

anything $< -- substitute a line from the standard input.

This redirection can be used in scripts, to get input from the user:

echo "How many do you want?"
set number = $<
echo "OK, I'll create $number of them."

And finally, the redirection of 'stderr':

program >& file -- redirect both errors and output to 'file'
program >>& file -- append both errors and output to 'file'

Normally when you redirect output from a Unix programs, error messages still

go to your terminal (or log file in a batch job). If you want the error messages to go to the same place as your output, you use the '>&' syntax. (And of course their counterparts '>&!' and '>>&!') Be careful of the syntax when using the '!' and '&' variations as misplacing the characters will produce random errors as these are job control characters in other contexts.

But wait! This only redirects 'stderr' into 'stdout', it doesn't let you independently redirect 'stderr' to a file other than where 'stdout' is going which was our original question. Well, there isn't any built-in syntax in generic 'csh' to do it. But you can redirect it, by doing the following:

(program > output.txt) >& error.txt

This redirects the 'stdout' of 'program' to 'output.txt' and then redirects any 'stdout' leftover (none of course) along with the error messages to 'error.txt' if you don't want to see the errors or want to analyze them (e.g. via 'grep') independently. Another useful special case of this syntax is:

(program > /dev/tty) >& /dev/null

This sends the output from 'program' to your terminal connection but redirects the errors to nowhere. (Really shouldn't ignore those errors, though. :-)

Shells other than 'csh' have different I/O redirection variations; in 'sh':

program optional-digit<&digit
program optional-digit>&digit

Lets you do redirection with file descriptors other than 'stdin', 'stdout' and 'stderr' or let you map those on top of each other. A commonly used example of this is found in the rc.* boot 'sh' scripts on /etc:

/usr/netware/etc/npsd && (echo -n ' npsd') >/dev/console 2>&1

This is effectively the 'sh' way of doing '>&' in 'csh' (since as you Unix junkies know, the '2' file descriptor is 'stderr' and '1' is 'stdout'.) The 'optional-digit>&digit' syntax is the general case of what 'csh' provides.

- Christopher